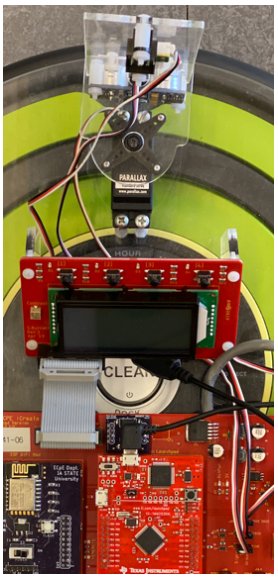
LAB 10

SERVO MOTOR POSITION CONTROL USING PULSE WIDTH MODULATION

INTRODUCTION

In this lab you will be implementing the control of a servo motor using pulse-width modulation (PWM). Understanding the operation of the servo and the GPTM timer in PWM mode will be important for your goals in this lab, which will also be important for your goals with the project.



About the Servo Motor

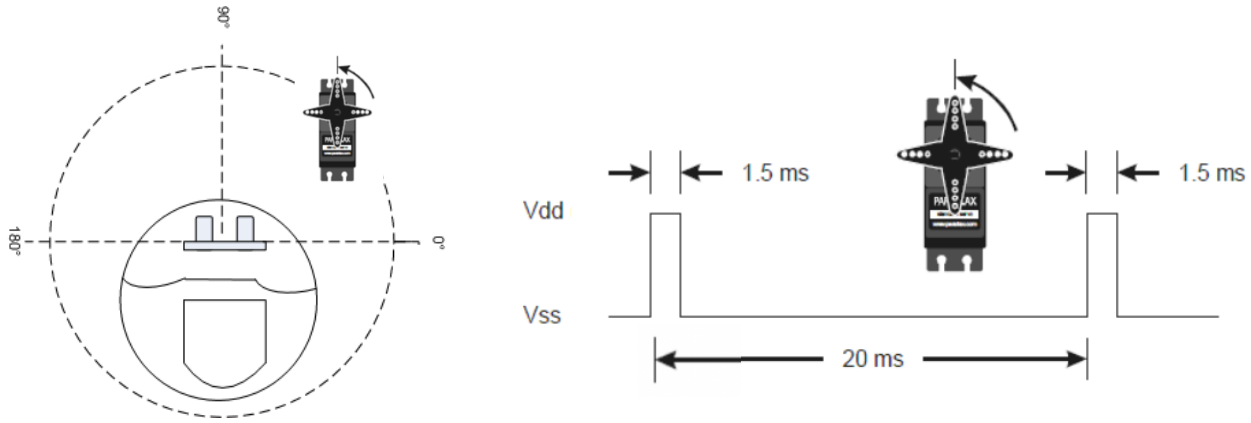
A servo motor is an electro-mechanical device with built-in position feedback. The device is commonly used in remote control devices and robotics and comes in two configurations: (1) one has a mechanical stop that restricts the servo to 180 degrees of motion, and (2) another allows 360 degrees of rotation. The servo that we will be using in lab has 180 degrees of rotation.

The main feature of a servo is the ability to position the device at an absolute position by sending the servo's control circuit a pulse train (periodic digital waveform of pulses) consisting of a positive pulse having a specific width.

Optionally, the servo has feedback circuitry that keeps the servo in a specified position. For example, if the servo were used to control a rudder or wing flap position, forces would push the controlled surface out of position unless there is feedback compensating for the forces. In our lab setup, the turret mounted on the servo will experience minimal forces that would move the turret out of position, so the feedback feature is not necessary.

Positioning the Servo Motor

The servo motor works by rotating to an absolute position using a control circuit driven by a pulse train that has a specified period and positive (high) pulse width. It typically expects to see a pulse every 20 milliseconds, and the length of the pulse determines how far and in which direction the motor turns. For example, if the pulse is high for **1.5 milliseconds**, then it will be at its **center position of 90 degrees**.



Because the servo must be positioned by generating a waveform that has events at precise times, the microcontroller has special hardware in the GPTM timer module, called output compare or pulse-width modulation, and this hardware is used to generate an output waveform having preset timing parameters. The output waveform is generated continuously and is called a pulse-width modulated signal, i.e., a periodic signal consisting of pulses of specific and varying duration. Modulation refers to modifying the pulse width.

- If the pulse is high for **1.5 milliseconds**, then it will be at its **center position of 90 degrees**.
- If the pulse is high for **1 millisecond**, then the servo will be positioned at **0 degrees (clockwise direction from center)**.
- If the pulse is high for **2 milliseconds**, it will be at **180 degrees (counterclockwise direction from center)**.

Pulse widths between **1 ms and 1.5 ms** will cause the motor to **rotate between 0 and 90 degrees**, and pulse widths between **1.5 ms and 2 ms** will cause the motor to **rotate between 90 and 180 degrees**.

A servo has a **linear relationship (y = mx+b) between pulse widths and servo position** in units of degrees. Pulse widths are specified as **timer count values** in the GPTM timer module. By determining the timer count values for 0 degrees and 180 degrees, a conversion factor can be used to convert a desired degree position into its corresponding counter value. The conversion factor is the ratio of the difference in counts over the range of motion in degrees.

****Note:** Testing will show that pulse widths translate to different absolute positions for each servo. Thus the conversion factor will be different for different bots. If you were experimenting with this hands-on in lab with your own bot, you would make adjustments based on the bot being used. There are various ways to implement adjustments, and all require observing the actual position of the servo compared to the desired position. For example if the desired position is 45 degrees, and the actual position observed is different (such as 50 degrees), your code should adjust the conversion factor so that it moves the servo to the desired position.

****Note 2**: When you used the pre-compiled Scan library, you followed a procedure to calibrate the servo given the Servo Calibration Reference Sheet. You found two calibration values. These

calibration values work with the Scan library functions but DO NOT WORK for other code. They are specific to the Scan library. You will need to implement your own calibration adjustments.

Recall that the General-Purpose Timer Module (GPTM) contains six timer blocks, Timer 0 through Timer 5. Each block can provide two 16-bit timers/counters: Timer A and Timer B. Thus, Timer A is a 16-bit timer, as is Timer B. Each GPTM block can work in one of several modes. In this lab we will be making use of **PWM Mode**, in order to generate a PWM output signal for the servo motor. The servo motor is wired to Timer 1B.

REFERENCE FILES

The following reference files will be used in this lab:

- lcd.c, program file containing various LCD functions
- lcd.h, header file for lcd.c
- timer.c, program file containing various wait commands
- timer.h, header file for timer.c
- Servo datasheet
- TI Tiva TM4C123G Microcontroller Datasheet
- TI TM4C123G Register Definitions C header file: REF_tm4c123gh6pm.h
- Cybot baseboard and LCD schematics: Cybot-Baseboard-LCD-Schematic.pdf
- GPIO and GPTM register lists and tables: GPIO-GPTM-registers-tables.pdf

There are no new code files to be included. You must include code files used in previous labs as needed.

In addition to previous files you may have included, you will need to write your own **servo.c** file and **servo.h** file and the associated functions for setting up and using PWM mode on the GPTM timer module. Separate functionalities should be in separate functions for good code quality and reusability. This means that in your servo.c file, you should write separate functions for initializing/configuring the PWM timer and positioning the servo. Remember to use good naming conventions for function names and variables. Minimally, we recommend defining the following functions:

```
void servo_init(void);
void servo_move(uint16_t degrees);
```

Your function prototypes may differ depending on the functionality you implement.

Note: No code has been provided for these functions.

PRELAB

See the prelab assignment in Canvas and submit it prior to or during lab.

Browse through the related readings and other resources provided in Canvas. Review the GPTM registers needed for this lab. You have previously used most of these registers in the PING lab, but you will set up PWM mode in this lab. A new register for this lab is the GPTMTnMATCHR match register (as well as the corresponding prescale match register). Refer to the Tiva datasheet, in particular, Figure 11-5 and section 11.4.5. See also sections 9.2.3.7 and 9.2.8.3 in the Bai book. Figure 9.5 in the Bai textbook is the same as Figure 11-5 in the datasheet. Refer to the lecture slides to get started with a high level sketch of your code.

STRUCTURED PAIRING

You should continue to use structured pairing in this lab. You may also start working with team members on this lab. If so, team members are encouraged to take specific roles, ask and prioritize questions, identify relevant information and tasks, and support coding, debugging and testing efforts.

PART 1: MOVING THE SERVO USING PWM

Develop an API that positions the servo at N degrees, where N is between 0 and 180 degrees. The position may not be true N degrees, however, that is okay for this part of the lab.

There is no way to know programmatically whether the servo has reached its new position. Positioning the servo 120 degrees away from its current position will take longer than positioning it 10 degrees away. This will require you to judge how long to wait prior to returning from the function and add a delay in the function. Keep in mind being overly conservative with a long delay will slow down servo movement sequences and the speed at which you can take sensor readings in new directions. However, being overly optimistic with a delay that is too short may result in sensor measurement errors, since you may collect sensor data before the sensor has reached its new position.

The Parallax servo datasheet does not specify the speed of operation of the motor. A typical hobby servo motor speed is 50 RPM (with no load and at full speed). A common metric is "time to 60 degrees." The Parallax servo is reportedly comparable to the Futaba S148, https://futabausa.com/product/s148/ . Keep in mind that the servo on the bot does have a load and is stopped to begin with, thus you will need to allow more time than specified. You do not need to calculate this precisely, but should have a reasonable delay.

In this lab, the PWM control signal for the servo is obtained using **Timer 1B**, which is connected to **GPIO Port B pin 5 (PB5)** and the "SERVO" header on the baseboard of the CyBot. **Timer1B is a 16-bit timer**, and in **PWM mode**, the **8-bit prescaler** register is used as an **extension of the timer**, creating a **24-bit timer** (most-significant bits are in the prescaler register). **The timer is configured as a 24-bit count-down counter in PWM mode. You will use it in periodic mode.**

1. The counter outputs a high PWM signal when it starts from a 24-bit start value loaded into the **GPTMTnILR** and **GPTMTnPR** load registers (most-significant bits are in the prescaler register). This is the **start value**.

2. It stays high until the current counter value equals a **match value** that is stored in the **GPTMTnMATCHR** and **GPTMTnPMR** match registers.

3. It then goes low and stays low until the counter reaches 0.

4. Once the counter reaches 0, it reloads the start value and repeats the cycle from step 1.

The **start value** can be considered to be the **period** of the PWM signal (in timer count values). The **match value** can be considered to be the **falling edge time** of the PWM signal, where the high pulse stops and the low pulse starts. So, **the match value is the width of the low pulse** for the PWM signal.

Let's state this again: **the match value is the width of the low pulse** for the PWM signal.

The **duty cycle** of the PWM signal is the **width of the high pulse divided by the period**. Formulas and an example of these calculations using the timer registers are given in section 9.2.8.3 of the Bai book (the example in the Bai book is for a 16-bit counter without prescaler).

You will need to initialize the timer module in PWM mode – see **section 11.4.5 PWM Mode in the Tiva Datasheet**). You will determine appropriate start and match values for the timer to generate a pulse train that positions the servo at N degrees. Figure 9.5 in the Bai book is the same as Figure 11-5 in the datasheet. Draw your own version of this figure for some value of N, e.g., let N=30 degrees. Show the start and match values in your figure.

Write a main program that moves the servo from N=90, then to N=30, then to N=150 and back to N=90, where it holds that position. Modify the match value for each N.

CHECKPOINT:

Demonstrate that you can move the servo to specific positions using PWM. The accuracy of the positions will not be evaluated, as you will calibrate the servo later. Use the PicoScope to display a waveform, and explain it in comparison with your drawing.

PART 2: USING PUSHBUTTONS TO MOVE THE SERVO

Expand your API with the following features.

1) Use pushbutton switches to adjust the match value to move the servo some number of degrees, as follows:
    a. SW1: Move the servo 1 degree in the current direction
    b. SW2: Move the servo 5 degrees in the current direction
    c. SW3: Switch between clockwise and counterclockwise movement of the servo
    d. SW4: Move the servo to 5 degrees, when in clockwise mode; move the servo to 175 degrees when in counterclockwise mode.
2) The servo should have an initial position of 90 degrees, and an initial direction of counterclockwise.
3) Display the match value and the current direction on the LCD.

4) Operate within the bounds of 0 to 180 degrees.

CHECKPOINT:

Demonstrate that pushbutton switches control the movement of the servo.

PART 3: ACCURATELY POSITION THE SERVO

Expand your API and user interface so that the program accurately positions the servo at any target position.

Use a protractor (provided or printed) to determine actual positions for counter values compared to specified angles. Calculate a conversion factor.

OPTIONAL: Implement user-assisted calibration for accurate positioning. Let the program start with a calibration phase in which the user is interacting with the program to confirm the actual position of the servo. The calibration phase will result in a conversion factor that computes accurate positions.

CHECKPOINT:

Demonstrate that the servo position is accurate.

PART 4: INTEGRATING YOUR GPTM FUNCTIONS

Congratulations! With the completion of all parts of this lab you will have implemented all of the primary features of the Scan library. You previously integrated your own UART and ADC functions into the Mission 2 code. Now you should integrate your GPTM functions for the PING sensor and servo. Your goal is to use your own developed code in the project.

CHECKPOINT:

The CyBot completes a mission with your GPTM code.

DEMONSTRATIONS:

1. **Functional demo of a lab milestone –** Specific milestone to demonstrate is Part 1.
2. **Debug demo using debugging tools to explain something about the internal workings of your system –** The TA will announce any specific debugging requirements at the start of lab; otherwise you will create your own debug demo based on your needs and interests in the lab.
3. **Q&A demo showing the ability to formulate and respond to questions –** This can be done in concert with the other demos.